Merchant Integration Guide

ACI Commerce Gateway™

Jan-2008, Release 3.2 Publication Number: EC-AB100-03

Contents

Preface

The ***ACI Commerce Gateway Merchant Integration Guide*** describes the use of the native interface to ACI Commerce Gateway and the use of the merchant plug-ins supported by the ACI Commerce Gateway product.

Audience

The ***ACI Commerce Gateway Merchant Integration Guide*** is mainly intended for merchant developers and integrators who wish to connect to ACI Commerce Gateway to send and receive transaction information.

Additional Documentation

The ACI Commerce Gateway documentation set is arranged so that each ACI Commerce Gateway manual presents a topic or group of related topics in detail. When one ACI Commerce Gateway manual presents a topic that has already been covered in detail in another ACI Commerce Gateway manual, the topic is summarized and the reader is directed to the other manual for additional information. Information has been arranged in this manner to be more efficient for readers who do not need the additional detail and at the same time provide the source for readers who require the additional information.

Software

This manual documents standard processing as of its publication date. Software that is not current and custom software modifications (CSMs) may result in processing that differs from the material presented in this manual. The customer is responsible for identifying and noting these changes.

Publication Identification

Three entries appearing at the bottom of each page uniquely identify this ACI Commerce Gateway publication. The publication number (for example, EC-AB100-03 for the ***ACI Commerce Gateway Merchant Integration Guide***) appears on every page to assist readers in identifying the manual from which a page of information was printed. The publication date (for example, Jan-2008 for January 2008) indicates the issue of the manual. The software release information (for example, R3.2 for release 3.2) specifies the software that the manual describes. This information matches the document information on the copyright page of the manual.

1: Coding to the ACI Commerce Gateway Native API

The native API (Application Program Interface) provides developers with a method of integrating batch and online processing of payment transactions into the merchant's application. The native API is a set of simple protocols for communicating with the ACI Commerce Gateway.

BatchPortal Protocol Specifications

The BatchPortal Plug-in transactions have the following characteristics:

Protocol	http
Port	443 Verisign 3.0 SSL Certificate
Target (action)	https://<host address>/<context>servlet/BatchPortalHTTPServlet
Method	POST
Content-Type	application/www-form-urlencoded or application/x-www-form-urlencoded - ENCTYPE="multipart/form-data" for Batch Action 1 (SSL File Upload)
ENCTYPE	"multipart/form-data" for Batch Action 1 (SSL File Upload)
Request Data Format	URL Encoded
Response Data Format	Single text string response delimited by colons
Encryption Level	SSL Version 3.0

Standard Transaction Example:

URL Encoded Data
id=TranPortal ID&password=password&batchaction=1&batchtrackid=unique tracking id

Multipart Form-Data
batchdata=batchdata type=file

Batch Request Record Format

Position	Data Value	Position	Data Value
1	action	11	trackid
2	card	12	UDF1
3	exp	13	UDF2
4	cvv2	14	UDF3
5	currencycode	15	UDF4
6	transid	16	UDF5
7	zip	17	CAVV
8	addr	18	XID
9	member	19	ECI
10	amt	Terminator	CRLF, CR, or LF

All fields must be delimited by a comma in the batch record. If the field is not used for a requested action, a comma must still be present. Do not use spaces for unused fields.

Batch Response Record Format

Position	Data Value	Position	Data Value
1	action	16	UDF5
2	card	17	CAVV
3	exp	18	XID
4	cvv2	19	ECI
5	currencycode	20	status
6	transid	21	error
7	zip	22	result
8	addr	23	auth
9	member	24	ref
10	amt	25	avr
11	trackid	26	date
12	UDF1	27	brand
13	UDF2	28	tranid
14	UDF3	Terminator	CRLF, CR, or LF
15	UDF4		

Transaction response data is appended to the end of the originating request record and returned as a single record in the batch output file.

Table of Batch Action Descriptions

Action	Function	Description
1	SSL File Upload	Performs an SSL File Upload of the Batch Transmit File. Query attempts can be made against a submitted batch to determine if the batch has finished processing or what percent of the batch has been processed.
2	SSL Batch Query and/or File Download	Performs an SSL Batch Query and/or File Download of the Batch Response File. The Batch Response File is sent after it is executed. If the Batch Response File is requested while executing, only the response data is sent.
3	SSL Cancellation	Performs an SSL cancellation indicating that the original Batch Transmit File should be terminated from its position in the Batch Server. Only the original ID can cancel the submitted batch. Once cancelled, Batch Action 2 (Batch Query/Receive) will transmit the results of the cancelled batch.
4	SSL Batch Query and/or File Download of the Authorized Batch Response File	Performs an SSL Batch Query and/or File Download of the Authorized Batch Response File. The Batch Response File containing only the authorized transactions is sent after it is executed. If the Batch Response File is requested while executing, only the response data is sent.
5	SSL Batch Query and/or File Download of the Denied Batch Response File	Performs an SSL Batch Query and/or File Download of the Denied Batch Response File. The Batch Response File containing only the denied transactions is sent after it is executed. If the Batch Response File is requested while executing, only the response data is sent.

Action	Function	Description
6	SSL Batch Query and/or File Download of the Rejected Batch Response File	Performs an SSL Batch Query and/or File Download of the Rejected Batch Response File. The Batch Response File containing only the rejected transactions is sent after it is executed. If the Batch Response File is requested while executing, only the response data is sent.

Request	Description	1	2	3	4	5	6
id	The TranPortal identification number issued by the ACI Commerce Gateway administrator and used to identify the merchant's terminal on which the transactions will be processed.	X	X	X	X	X	X
password	The TranPortal Password issued by the ACI Commerce Gateway administrator and used to authenticate the merchant on transaction requests.	X	X	X	X	X	X
batchaction	Valid BatchPortal actions are identified below. Use the numerical format only. 1 = Send the batch file for processing. 2 = Query/Receive the batch response file. 3 = Send the batch cancellation request.	X	X	X	X	X	X
batchtrackid	A unique tracking ID provided by the merchant's system that is stored with the batch file transaction. This ID must be unique and can be used to query batch status, cancel a processing batch file, or retrieve batch response files. alphanumeric - no spaces	X	X	X	X	X	X
batchdata	File created by merchant's system.	X					
filename	The name of the file as provided by the merchant or as derived by the plug-in.	X					

Note: The response is a colon delimited string (e.g batchtrackid:batchid:filesize).

Response	Description	1	2	3	4	5	6
!ERROR!errortext -or- batchtrackid	Format of error response. No other fields are returned in an error condition. The unique tracking ID echoed from the original request.	X	X	X	X	X	X
batchid	A unique number returned by the ACI Commerce Gateway system identifying the batch.	X	X	X	X	X	X
filesize	An integer value returned by the ACI Commerce Gateway representing the size of the batch file received (in bytes).	X	X	X	X	X	X
batchstatus	An integer value returned by the ACI Commerce Gateway representing the status of the batch submitted. Valid values: 0 = Waiting In Queue 1 = Processing 2 = Finished 3 = Cancelled 4 = Stopped 5 = Error Occurred		X				
batchstatuspercent	An integer value returned by the ACI Commerce Gateway representing the percent complete of the batch submitted.		X				
batchoutdata	The resulting batch response file.		X				

ACI Worldwide, Inc.

2: Coding to the ACI Commerce Gateway Merchant Plug-ins

The Universal Plug-in is a single plug-in that supports multiple transaction types with the ACI Commerce Gateway. The Universal Plug-in provides the necessary framework to send and receive messages simultaneously using HTTP and converts each supported transaction to an accessible interface. This abstraction standardizes the transaction handling across all transactions, except for e24BatchPipe (it uses a separate ISO standard to communicate and is not currently supported via the Universal Plug-in) and improves the turnaround time taken to customize these transactions for special customer requests.

The definition files are delivered within the config directory in a subdirectory called UniversalPlugin and take the form “TRAN_” + the transaction name + “VER_” + the version of the transaction definition + “.xml” (i.e., TRAN_ TranPortal_VER_1.xml). The format of these files is standard XML, but the contents of the XML are currently proprietary and not open to customer manipulation. In the future, customer manipulation may be offered.

BatchPortal C++ Plug-in

The C++ version of the e24BatchPipe plug-in is designed for programmers using Visual C++. The component is based on a pure virtual interface to the object (Ie24BatchPipe), obtained by calling a single factory function exported by the DLL (Createe24BatchPipe). The component has the ability to deliver asynchronous status messages during the transaction. Any object that wants to receive these notifications must implement the Ie24BatchPipeStatus interface.

File name: e24BatchPipe

Location: Plug-in Downloads

Refer to the "C++ Components" topic for more information.

Normal Response Conditions

- Normal transaction processing
- PerformTransaction returns 0
- Merchant Web software site should not need to initiate any special processing

ACI Commerce Gateway Error Response Conditions

- Error transaction processing
- PerformTransaction returns -1
- GetErrorMsg should return text
- The ACI Commerce Gateway encountered an error with either the data validation, database access, or system-related issues
- Merchant Web software site should not need to initiate any special processing

Plug-in Error Response Conditions

- Error DLL processing
- PerformTransaction returns -1
- GetErrorMsg should return text
- e24 dll plug-ins utilize the Windows HTTP Services (WinHTTP)

- Merchant Web software may need to format reversal transaction for:

  ERROR_WINHTTP_TIMEOUT and ERROR_WINHTTP_OPERATION_CANCELLED since it is possible that the request message was transmitted and the response results are unknown.

- e24BatchPipe maps the most common WinHTTP errors into text

  ErrorMsg Text Common WinHTTP Errors

  “Connection timed out”, // ERROR_WINHTTP_TIMEOUT

  “Internal Error”, // ERROR_WINHTTP_INTERNAL_ERROR

  “Invalid URL”, // ERROR_WINHTTP_INVALID_URL

  “Name not resolved”, // ERROR_WINHTTP_NAME_NOT_RESOLVED

  “Cannot Connect”, // ERROR_WINHTTP_CANNOT_CONNECT

  “Connection Aborted”, // ERROR_WINHTTP_OPERATION_CANCELLED

- All other WinHTTP errors will result in the following ErrorMsg:

  “Connection Failed with error <WinHTTP Error>”

WinHTTP Errors: http://msdn2.microsoft.com/en-us/library/aa384110.aspx

BatchPortal Java Class Object

The Java Class Object version of the e24BatchPipe plug-in can be used in a variety of developmental environments for a wide range of desktop to Web-based applications that are platform independent. The e24BatchPipe Java Class Object is a royalty-free, distributable component that any developer can use to enable their Internet e-commerce applications or websites for BatchPortal transactions.

Refer to the "Java Components" topic for more information.

Universal Plug-in Installation

The Universal Plug-in defines fields for both requests and responses for a number of transactions into XML definition files. These transaction definition files define both server and client side requirements. The Universal Plug-in requires these definition files at both the client and server to function properly. The files are automatically distributed to the client via inclusion into the "secure resource" file already in use. These definition files must be installed with the ACI Commerce Gateway in a directory and then the config.property "universal.plugin.xmlfiles. location" must point to where they reside. Once these files are installed into ACI Commerce Gateway in this manner, regenerating the merchant resource files results in a slightly larger file which contains these transaction definitions for use with the Universal Plug-in at the merchant client site as well as allowing the Universal Plug-in servlets the necessary definitions to process the contained transaction types.

Note: The Universal Plug-in requires a merchant resource be generated with the transaction definition files within it for any communication to occur.

Universal Plug-in Supported Transactions

The following transactions are supported by the Universal Plug-in.

Transaction Name	Transaction Description
CardManagement	This transaction makes available balance inquiry, replenishment and void replenishment transactions for supporting stored value cards. **Note:** Formerly supported by the e24CardPipe plug-in.

Transaction Name	Transaction Description
MPIPayerAuthentication	This transaction makes available communication to an Access Control Server (ACS) via the ACI Commerce Gateway for continuing the 3-D Secure transactions started by MPIVerifyEnrollment by submitting the 3-D Secure "PARes" message returned from the ACS to the ACI Commerce Gateway along with the payment ID that was returned after the MPIVerifyEnrollment. Transactions initiated through the MPIVerifyEnrollment transaction are tied to orders but do not flow through the hosted payment page. It is up to the merchant to collect card information prior to calling the MPIVerifyEnrollment procedure as the generated "PAReq" message has a lifespan limited by 3-D Secure's ACS. **Note:** Formerly known as the performPATransaction() method of the e24VBVPlugin.
MPIVerifyEnrollment	This transaction makes available communication to an Access Control Server (ACS) via the ACI Commerce Gateway for performing the 3-D Secure "VEReq/VERes" transactions as well as generating the 3-D Secure "PAReq" request necessary to start consumers with their authentication process with the ACS. Transactions initiated through the MPIVerifyEnrollment transaction are tied to orders but do not flow through the hosted payment page. It is up to the merchant to collect card information prior to calling the MPIVerifyEnrollment procedure as the generated "PAReq" message has a lifespan limited by 3-D Secure's ACS. **Note:** Formerly known as the performVETransaction() method of the e24VBVPlugin.
PaymentInit	This transaction makes available Authorization, Capture, Credit, Purchase, Query, Void Authorization, Void Capture, Debit, and Void Purchase transactions for credit cards, debit cards, and VPAS cards. Transactions initiated through the PaymentInit transaction are tied to orders and are expected (for authorizations and purchases) to flow through the hosted payment page. **Note:** Formerly known as the performPaymentInitialization () method of the e24PaymentPipe plug-in.

Transaction Name	Transaction Description
PaymentTran	This transaction makes available Authorization, Capture, Credit, Purchase, Query, Void Authorization, Void Capture, Debit, and Void Purchase transactions for credit cards, debit cards, and VPAS cards. Transactions initiated through the PaymentInit transaction are tied to orders and are expected (for authorizations and purchases) to flow through the hosted payment page. **Note:** Formerly known as the performTransaction() method of the e24PaymentPipe plug-in.
TranPortal	This transaction makes available Authorization, Capture, Credit, Purchase, Query, Void Authorization, Void Capture, Debit, and Void Purchase transactions for credit cards, debit cards, and VPAS cards. Transactions initiated through the TranPortal transaction are not tied to orders and do not flow through the hosted payment page. **Note:** Formerly known as the performTransaction() of the e24TranPipe plug-in.

Universal Plug-in Definition Files

It is important to understand that the field definitions for the requests and responses to individual transactions may change; however, the general technology for determining the fields that are available after these changes occur remains the same.

The Universal Plug-in definition files are in XML format with the schema shown on the following page. This schema shows that the definition file contains a root transaction tag. This tag currently contains a single request tag followed by two response tags (one for valid, one for error), each containing field tags which describe the fields that make up each request or response. Each of the tags has some required attributes which define those tags. The transaction tag must have a name and version while class and method are only required on the server and not for the plug-ins. The response tags must have a type attribute which is set to either valid or error. Each field tag must have an id, refID, and type attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XML Spy v4.3 U (http://www.xmlspy.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
       <xs:element name="field">
            <xs:complexType>
            <xs:attribute name="id"  type="xs:string" use="required"/>
            <xs:attribute name="refID"  type="xs:string" use="required"/>
            <xs:attribute name="type"  type="xs:string" use="required"/>
            </xs:complexType>
       </xs:element>
       <xs:element name="transaction">
            <xs:complexType>
            <xs:sequence>
            <xs:element name="request">
            <xs:complexType>
            <xs:sequence>
            <xs:element ref="field" maxOccurs="unbounded"/>
            </xs:sequence>
            </xs:complexType>
            </xs:element>
            <xs:element name="response" maxOccurs="unbounded">
            <xs:complexType>
            <xs:sequence>
            <xs:element ref="field" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="type" use="required">
            <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="error"/>
            <xs:enumeration value="valid"/>
            </xs:restriction>
            </xs:simpleType>
            </xs:attribute>
            </xs:complexType>
            </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="version" type="xs:string" use="required"/>
            <xs:attribute name="class" type="xs:string" use="required"/>
            <xs:attribute name="method" type="xs:string" use="required"/>
            </xs:complexType>
       </xs:element>
</xs:schema>
```

Universal Protocol Specifications

The Universal Plug-in transactions have the following characteristics:

Protocol	http
Port	443 Verisign 3.0 SSL Certificate
Target	https://<host address:port>/<context>/servlet/UniversalXMLServlet
Method	POST
Content-Type	application/www-form-urlencoded or application/x-www-form-urlencoded
Request Data Format	URL Encoded
Response Data Format	Single text string response delimited by colons
Encryption Level	SSL Version 3.0

Exceptions: The only transmissions that are not in XML are the messages to and from the Merchant Notification Service at the end of the PaymentInit Universal Plug-in transaction flow. The Merchant Notification transmissions are expected to be in HTML form data format and are posted from the Merchant Notification to the merchant response URL. The merchant response URL must respond with a redirection URL that the consumer will be sent to via a ***get*** method. For this reason, it is recommended that no private data be used on this merchant-supplied redirection URL.

Interpreting the available fields for a transaction's response and request is possible by reading the definition file. The field tags within request and response portions of the definition files specify the XML which will be generated by the Universal Plug-in to send the request to and receive responses from the ACI Commerce Gateway. For example, the id attribute represents the tag name to be generated. For instance, given the definition:

```
<?xml version="1.0" encoding="UTF-8"?>
      <transaction name="TranPortal" class="com.aciworldwide.commerce.gateway.
PortalTransaction" method="processTransactionExternal" version="1">
           <request action="4">
           <field id="id" refID="tran_portal_id_str" type="String"/>
           <field id="password" refID="tran_portal_pwd_str" type="String"/>
           <field id="passwordhash" refID="tran_portal_pwd_hash_str" type="String"/>
           <field id="card" refID="card_number_str" type="String" />
           <field id="cvv2" refID="card_verification_code_str" type="String" />
           <field id="expyear" refID="card_expiration_year_int" type="int" />
           <field id="expmonth" refID="card_expiration_month_int" type="int" />
           <field id="expday" refID="card_expiration_day_int" type="int" />
           <field id="action" refID="tran_action_int" type="int" />
           <field id="type" refID="payment_instrument_str" type="String" />
           <field id="transid" refID="orig_transaction_id_lng" type="long"/>
           <field id="zip" refID="card_postal_code_str" type="String" />
           <field id="addr" refID="card_address_str" type="String" />
           <field id="member" refID="card_name_str" type="String" />
           <field id="amt" refID="tran_amount_dbl" type="double" />
           <field id="currencycode" refID="currency_code_str" type="String" />
           <field id="trackid" refID="track_id_str" type="String" />
           <field id="udf1" refID="user_field_1_str" type="String" />
           <field id="udf2" refID="user_field_2_str" type="String" />
           <field id="udf3" refID="user_field_3_str" type="String" />
           <field id="udf4" refID="user_field_4_str" type="String" />
           <field id="udf5" refID="user_field_5_str" type="String" />
           <field id="cardcert" refID="cardholder_certificate_byte" type="byte"/>
           <field id="merchantcert" refID="merchant_certificate_byte" type="byte"/>
           <field id="cavv" refID="authentication_token_encoded_str" type="String"/>
           <field id="eci" refID="electronic_commerce_indicator_str" type="String"/>
           <field id="xid" refID="set_gateway_xid_byte" type="byte"/>
           </request>
           <response type="error">
           <field id="error_code_tag" refID="error_code_str" type="String"/>
           <field id="error_service_tag" refID="error_service_str" type="String"/>
           <field id="error_text" refID="error_text_str" type="String"/>
           </response>
           <response type="valid">
           <field id="result" refID="result_code_str" type="String"/>
           <field id="auth" refID="auth_code_str" type="String"/>
           <field id="ref" refID="tran_ref_str" type="String"/>
           <field id="cardbalance" refID="card_balance_str" type="String"/>
           <field id="avr" refID="address_verification_code_str" type="String"/>
           <field id="postdate" refID="post_date_str" type="String"/>
           <field id="tranid" refID="transaction_id_lng" type="long"/>
```

```
            <field id="trackid" refID="track_id_str" type="String"/>
            <field id="udf1" refID="user_field_1_str" type="String"/>
            <field id="udf2" refID="user_field_2_str" type="String"/>
            <field id="udf3" refID="user_field_3_str" type="String"/>
            <field id="udf4" refID="user_field_4_str" type="String"/>
            <field id="udf5" refID="user_field_5_str" type="String"/>
            <field id="udf5" refID="user_field_6_str" type="String"/>
            <field id="responsecode" refID="auth_response_code_str" type="String"/>
            </response>
      </transaction>
```

It is possible to determine the expected raw interface format expected by the ACI Commerce Gateway's Universal Servlet interface as shown below:

```
<transaction name="TranPortal"  version="1">
      <request>
            <id></id>
            <password></password>
            <passwordhash></passwordhash>
            <card></card>
            <cvv2></cvv2>
            <expyear></expyear>
            <expmonth></expmonth>
            <expday/>
            <action></action>
            <type></type>
            <transid></transid>
            <zip></zip>
            <addr></addr>
            <member></member>
            <amt></amt>
            <currencycode></currencycode>
            <trackid></trackid>
            <udf1></udf1>
            <udf2></udf2>
            <udf3></udf3>
            <udf4></udf4>
            <udf5></udf5>
            <cardcert></cardcert>
            <merchantcert></merchantcert>
            <cavv></cavv>
            <eci></eci>
            <xid></xid>
            </request>
</transaction>
```

Note: The above XML is in a user-friendly format; however, the actual XML generated by the Universal Plug-in and Servlet does not contain the carriage return, line feed, and tabular formatting in order to save transmission space.

The format does not currently specify which fields are required nor which fields are optional. This may become available in future versions of the Universal Servlet.

The "Universal Plug-in Required Fields" topic describes the meaning of each field in the current transaction definitions and whether it is required or not.

Universal Plug-in Programming Interface

With the migration to the Universal Plug-in, ACI Commerce Gateway has changed its programming language offerings. There are COM+ and Java versions available of the Universal Plug-in. Consideration was taken to try to keep the interfaces to these plug-ins virtually identical in order to allow easy explanation and usage of each of these programming language versions.

In every language supported (COM+, Java, and PHP), each transaction contains the following:

- A ***get*** method for getting values received by the plug-in. The values are returned as strings.
- A ***set*** method for setting values to be transmitted by the plug-in. The key and values to set are returned as strings.
- A ***setTransactionType*** method for setting the transaction name being sent.
- A ***setVersion*** method for setting the transaction version being sent.
- A ***setTerminalAlias*** method for setting the terminal alias to find in the merchant resource for establishing communication with the ACI Commerce Gateway.
- A ***setResourcePath*** method for setting the physical path to the merchant resource for establishing communication with the ACI Commerce Gateway.
- Only one execution method called ***performTransaction*** after which the merchant code should check the ***getErrorText*** method to confirm that there are no errors.

- A ***getErrorText*** method for determining whether performTransaction succeeded in its parsing, processing, and transmission to the ACI Commerce Gateway. This is not the same as using the ***get*** method for "error," which returns the server's reply if it was an error.

An example of using these methods to perform a TranPortal transaction is shown on the following pages for each language. Different languages use different syntax for class method invocation but the method names themselves are the same.

TranPortal Transaction using the COM+ Universal Plug-in in Visual Basic

```
'  After Adding a reference to: UniversalPlugin 1.0 Type Library
Dim plug As UniversalPlugin.CUniversalPluginCtl
Set plug = New UniversalPlugin.CUniversalPluginCtl

' ------------------------------------------------------
'   Set up the proper transaction
' ------------------------------------------------------
plug.setResourcePath Me.txtResourcePath
plug.setTerminalAlias Me.txtTerminalAlias
plug.setTransaction Me.cmbTransaction
plug.setVersion Me.txtVersion

' ------------------------------------------------------
'   Set up the transaction data
' ------------------------------------------------------
plug.Set "card", "value for card"
plug.Set "cvv2", "value for cvv2"
plug.Set "expyear", "value for expyear"
plug.Set "expmonth", "value for expmonth"
plug.Set "expday", "value for expday"
plug.Set "action", "value for action"
plug.Set "type", "value for type"
plug.Set "transid", "value for transid"
plug.Set "zip", "value for zip"
plug.Set "addr", "value for addr"
plug.Set "member", "value for member"
plug.Set "amt", "value for amt"
plug.Set "currencycode", "value for currencycode"
plug.Set "trackid", "value for trackid"
plug.Set "udf1", "value for udf1"
plug.Set "udf2", "value for udf2"
plug.Set "udf3", "value for udf3"
plug.Set "udf4", "value for udf4"
plug.Set "udf5", "value for udf5"

' ------------------------------------------------------
'     Sent the transaction.
' ------------------------------------------------------
plug.performTransaction
```

```
ErrorMessage = plug.getErrorText

If (Len(ErrorMessage) = 0) Then
    resultValue = plug.get("result")
    authValue = plug.get("auth")
    refValue = plug.get("ref")
    cardbalanceValue = plug.get("cardbalance")
    avrValue = plug.get("avr")
    postdateValue = plug.get("postdate")
    tranidValue = plug.get("tranid")
    trackidValue = plug.get("trackid")
    udf1Value = plug.get("udf1")
    udf2Value = plug.get("udf2")
    udf3Value = plug.get("udf3")
    udf4Value = plug.get("udf4")
    udf5Value = plug.get("udf5")
    responsecodeValue = plug.get("responsecode")
End If
```

TranPortal Transaction using the COM+ Universal Plug-in in ASP

```
' ------------------------------------------------------
'  After Adding a reference to: UniversalPlugin
' ------------------------------------------------------
Dim plug
Set plug = Server.CreateObject("UniversalPlugin.UniversalPluginCtl")

' ------------------------------------------------------
'   Set up the proper transaction
' ------------------------------------------------------
plug.setResourcePath  resourcePath
plug.setTerminalAlias terminalAlias
plug.setTransaction   "TranPortal"
plug.setVersion       "1"

' ------------------------------------------------------
'   Set up the transaction data
' ------------------------------------------------------
plug.Set set "card", "value for card"
plug.Set set "cvv2", "value for cvv2"
plug.Set "expyear", "value for expyear"
plug.Set "expmonth", "value for expmonth"
plug.Set "expday", "value for expday"
plug.Set "action", "value for action"
plug.Set "type", "value for type"
plug.Set "transid", "value for transid"
```

```
    plug.Set "zip", "value for zip"
    plug.Set "addr", "value for addr"
    plug.Set "member", "value for member"
    plug.Set "amt", "value for amt"
    plug.Set "currencycode", "value for currencycode"
    plug.Set "trackid", "value for trackid"
    plug.Set "udf1", "value for udf1"
    plug.Set "udf2", "value for udf2"
    plug.Set "udf3", "value for udf3"
    plug.Set "udf4", "value for udf4"
    plug.Set "udf5", "value for udf5"

    ' ------------------------------------------------------
    '     Sent the transaction.
    ' ------------------------------------------------------
    plug.performTransaction

    ErrorMessage = plug.getErrorText

    If (Len(ErrorMessage) = 0) Then
        resultValue = plug.get("result")
        authValue = plug.get("auth")
        refValue = plug.get("ref")
        cardbalanceValue = plug.get("cardbalance")
        avrValue = plug.get("avr")
        postdateValue = plug.get("postdate")
        tranidValue = plug.get("tranid")
        trackidValue = plug.get("trackid")
        udf1Value = plug.get("udf1")
        udf2Value = plug.get("udf2")
        udf3Value = plug.get("udf3")
        udf4Value = plug.get("udf4")
        udf5Value = plug.get("udf5")
        responsecodeValue = plug.get("responsecode")
    End If
```

TranPortal Transaction in Java

```
import com.aciworldwide.commerce.gateway.plugins.UniversalPlugin;

…

        UniversalPlugin plug = new UniversalPlugin();

        // ------------------------------------------------------
        //   Set up the proper transaction
        // ------------------------------------------------------
        plug.setResourcePath(resourcePath);
```

```
plug.setTerminalAlias(terminalAlias);
plug.setTransactionType("TranPortal");
plug.setVersion("1");

// ---------------------------------------------------
//   Set up the transaction data
// ---------------------------------------------------
plug.set("card", "value for card");
plug.set("cvv2", "value for cvv2");
plug.set("expyear", "value for expyear");
plug.set("expmonth", "value for expmonth");
plug.set("expday", "value for expday");
plug.set("action", "value for action");
plug.set("type", "value for type");
plug.set("transid", "value for transid");
plug.set("zip", "value for zip");
plug.set("addr", "value for addr");
plug.set("member", "value for member");
plug.set("amt", "value for amt");
plug.set("currencycode", "value for currencycode");
plug.set("trackid", "value for trackid");
plug.set("udf1", "value for udf1");
plug.set("udf2", "value for udf2");
plug.set("udf3", "value for udf3");
plug.set("udf4", "value for udf4");
plug.set("udf5", "value for udf5");

//  ---------------------------------------------------
//      Sent the transaction.
//  ---------------------------------------------------
plug.performTransaction();

String errorMessage = plug.getErrorText();

if (errorMessage.length() ==  0) {
    resultValue = plug.get("result");
    authValue = plug.get("auth");
    refValue = plug.get("ref");
    cardbalanceValue = plug.get("cardbalance");
    avrValue = plug.get("avr");
    postdateValue = plug.get("postdate");
    tranidValue = plug.get("tranid");
    trackidValue = plug.get("trackid");
    udf1Value = plug.get("udf1");
    udf2Value = plug.get("udf2");
    udf3Value = plug.get("udf3");
    udf4Value = plug.get("udf4");
    udf5Value = plug.get("udf5");
    responsecodeValue = plug.get("responsecode");
}
```

TranPortal Transaction in PHP

```
require_once "../Universal/UniversalPlugin.php";

$CGPipe = new UniversalPlugin($debug);

// ------------------------------------------------------
//   Set up the proper transaction
// ------------------------------------------------------
$CGPipe->setResourcePath($resourcePath);
$CGPipe->setTerminalAlias($terminalAlias);
$CGPipe->setTransactionType("TranPortal");
$CGPipe->setVersion("1");

// ------------------------------------------------------
//   Set up the transaction data
// ------------------------------------------------------
$CGPipe->set("card", "value for card");
$CGPipe->set("cvv2", "value for cvv2");
$CGPipe->set("expyear", "value for expyear");
$CGPipe->set("expmonth", "value for expmonth");
$CGPipe->set("expday", "value for expday");
$CGPipe->set("action", "value for action");
$CGPipe->set("type", "value for type");
$CGPipe->set("transid", "value for transid");
$CGPipe->set("zip", "value for zip");
$CGPipe->set("addr", "value for addr");
$CGPipe->set("member", "value for member");
$CGPipe->set("amt", "value for amt");
$CGPipe->set("currencycode", "value for currencycode");
$CGPipe->set("trackid", "value for trackid");
$CGPipe->set("udf1", "value for udf1");
$CGPipe->set("udf2", "value for udf2");
$CGPipe->set("udf3", "value for udf3");
$CGPipe->set("udf4", "value for udf4");
$CGPipe->set("udf5", "value for udf5");

//  ------------------------------------------------------
//      Sent the transaction.
//  ------------------------------------------------------
$CGPipe->performTransaction();

$errorMessage = $CGPipe->getErrorText();

if (strlen($errorMessage) ==  0) {
    $resultValue = $CGPipe->get("result");
    $authValue = $CGPipe->get("auth");
    $refValue = $CGPipe->get("ref");
    $cardbalanceValue = $CGPipe->get("cardbalance");
    $avrValue = $CGPipe->get("avr");
    $postdateValue = $CGPipe->get("postdate");
    $tranidValue = $CGPipe->get("tranid");
    $trackidValue = $CGPipe->get("trackid");
    $udf1Value = $CGPipe->get("udf1");
    $udf2Value = $CGPipe->get("udf2");
```

```
        $udf3Value = $CGPipe->get("udf3");
        $udf4Value = $CGPipe->get("udf4");
        $udf5Value = $CGPipe->get("udf5");
        $responsecodeValue = $CGPipe->get("responsecode");
    }
```

Universal Plug-in Transaction Flow

The Universal Plug-in works in conjunction with the ACI Commerce Gateway to support the following functionality:

- Simple transactions that do not make up an order
- Payments which allow both the creation of linked transactions to a single order and also allow use of the ACI Commerce Gateway hosted payment page
- Merchant Payment Interface (MPI) transactions which use ACI Commerce Gateway to format and process 3-D Secure transactions

Each of these capabilities requires at least one or two actual Universal Plug-in transactions to complete.

Simple Transactions

The ***TranPortal*** transaction is used to initiate and complete simple financial transactions. ***TranPortal*** transactions are separate from one another and are only linked by transaction/original transaction IDs. The original transaction ID is used for subsequent transactions on an authorization or purchase financial transaction. Refer to "3-D Secure Simple Transaction Flow" for a visual representation of this flow.

The ***CardManagement*** transaction is used to manage stored value cards only. The transaction provides balance inquiry, replenishment, and void replenishment transactions for stored value cards.

Payments

The ***PaymentInit*** and ***PaymentTran*** transactions are used as a pair to create payment transactions within the ACI Commerce Gateway. Each transaction is complete by itself, but provides a way for merchants to manage orders. Refer to "3-D Secure Payment Transaction Flow" for a visual representation of this flow.

The ***PaymentInit*** transaction initializes payments through the ACI Commerce Gateway hosted payment page to acquire the consumer's card details as well as automatically redirect to 3-D Secure if the card is enrolled, whereas the ***PaymentTran*** transaction allows subsequent transactions on those payment transactions created through ***PaymentInit***.

The ***PaymentInit*** transaction initializes either a purchase or authorization within the ACI Commerce Gateway. The transaction requires the merchant to provide the ACI Commerce Gateway with the necessary response URL at the merchant to redirect to in case an error occurs and also if everything is proceeding successfully. If the reply contains no errors, a payment page URL and payment ID is provided in order for the merchant to redirect the customer on to the ACI Commerce Gateway's hosted payment page. If the card is determined to be enrolled in 3-D Secure after instrument details are entered and confirmed on the hosted payment page, the customer will be redirected to the ACS before returning to the response URL sent in on the request. The merchant's response URL, whether a servlet or JSP page, is required to read the reply and store it before replying with a receipt URL to which ACI Commerce Gateway will redirect the consumer.

The ***PaymentTran*** transaction requires that the merchant have already acquired either a payment ID from a previous payment transaction or the card details to create an ACI Commerce Gateway transaction. The ***PaymentTran*** can be used to perform subsequent transactions against any other payment transaction given its payment ID.

Merchant Payment Interface (MPI) to 3-D Secure

The ***MPIVerifyEnrollment*** and ***MPIPayerAuthentication*** Universal Plug-in transactions are used as a pair to provide enrollment verification and consumer authorization with a 3-D Secure Access Control Server (ACS) through the ACI Commerce Gateway. Refer to "3-D Secure Merchant Payer Interface (MPI) Transaction Flow" for a visual representation of this flow.

The ***MPIVerifyEnrollment*** transaction requests the ACS to verify the enrollment status of a submitted card number.

If the card is not enrolled, by default, the ACI Commerce Gateway processes the transaction as a credit card transaction, and the returned fields will be the same as those returned by the ***PaymentTran*** Universal Plug-in transaction.

If the card is enrolled, the consumer must authenticate themselves to the ACS before the transaction can continue to be processed as a 3-D Secure transaction. The return values for the PaymentTran transaction include:

- The URL for the ACS's authorization page
- A properly formatted PAReq 3-D Secure message
- A payment ID (returned in the MD field) assigned to the transaction by the ACI Commerce Gateway

The merchant must also provide a URL for the ACS to redirect the user to after authentication. That terminal URL (called the TermURL) will be used by the ACS after authentication is complete. The merchant redirects the consumers browser to the ACS page returned, submitting the following required HTML form values: "PAReq" " (containing the 3-D Secure Payer Authentication Request return value from the MPIVerifyEnrollment call), "TermURL" and "MD" (containing the payment ID return value from the MPIVerifyEnrollment call).

The page or servlet given in the Terminal URL continues processing the transaction by sending the PARes and payment ID (returns MD, which stands for Merchant Digest) values that were posted to it through the consumer redirection from the ACS.

The MPIPaymentAuthentication transaction requires the above returned PARes as well as the payment ID (returned in the MD form field) and continues processing the transaction. If the transaction is successful, the returned field will be the same as those returned by the ***PaymentTran*** Universal Plug-in transaction.

Universal Plug-in Required Fields

All Universal Plug-in transactions require, at a minimum, four common values. The Universal Plug-in provides four unique methods for setting these values:

- The ***setTransactionType*** method for setting the transaction name being sent.
- The ***setVersion*** method for setting the transaction version being sent.
- The ***setTerminalAlias*** method for setting the terminal alias to find in the merchant resource for establishing communication with the ACI Commerce Gateway.
- The ***setResourcePath*** method for setting the physical path to the merchant resource for establishing communication with the ACI Commerce Gateway.

These four values are used to locate the secure resource, open it, determine the transaction and transaction version to use to communicate to the ACI Commerce Gateway, and finally, determine the connection credentials (the transaction portal id and password or passwordhash) to allow communicating through a certain terminal. Aside from these values retrieved by these methods, certain values are required for each of the Universal Plug-in transactions.

The following table illustrates the required fields for each Universal Plug-in transaction:

Transaction	Required Request Fields	
CardManagement	action	currencycode
	card	member
	expyear	amt
	expmonth	cvv2 (if the brand requires it)
MPIPayerAuthentication	pares	
	payementid	
MPIVerifyEnrollment	action	member
	card	amt
	type	currencycode
	expyear	cvv2
	expmonth	
PaymentInit	action	responseurl
	amt	errorurl
	currencycode	
PaymentTran	action	paymentid
	amt	transid **
	currencycode	
TranPortal	action	currencycode
	card *	member
	type	amt
	expyear *	cvv2 *
	expmonth *	transid **

* This field is not required on subsequent transactions.

** This field is only required on subsequent transactions.

Universal Plug-in Transaction Field Values

The following table describes the fields that can be sent or received in Universal Plug-in transactions. The field names are not only the valid values for the Universal Plug-ins' ***set*** and ***get*** methods but also represent the raw XML tag names used by the raw interface.

Field Name	Description
action	Indicates the desired action to perform at the ACI Commerce Gateway for the requested transaction. Valid values are 1 through 12. Required. Following are valid values for the TranPortal, PaymentInit, PaymentTran, and MPIVerifyEnrollment transactions: 1 - Purchase 2 - Credit 3 - Void Purchase 4 - Authorization 5 - Capture 6 - Void Credit 7 - Void Capture 8 - Query 9 - Void Authorization Following are valid values for the CardManagement transactions: 10 - Balance Inquiry 11 - Replenishment 12 - Void Replenishment
addr	The address of the card billing address. Review the config.properties file for valid character limitations under the hack character id *hc.bad_address.*
amt	The amount of the transaction. Review the config.properties file for valid character limitations under the hack character id *hc. bad_currency.*
auth	The authentication code value returned from the host.

Field Name	Description
avr	The address verification result code value returned from the host. This result code is a single letter indicating how closely the submitted address information matched the cards billing address information. Valid values: A - Address matches. E - Address match error. N - No address match. R - AVS not available. S - Service not supported. U - Address match not capable. W - 9-digit ZIP Code matches. X - Address and 9-digit ZIP Code matches. Y - Address and 5-digit ZIP Code matches. Z - 5-digit ZIP Code matches. 0 - Address could not be verified.
card	The card number to use for the transaction.
cardbalance	The monetary balance left on a card.
cardcert	Currently unused.
cavv	Card Authentication Verification Value used by a 3-D Secure Access Control Server (ACS) to report successful authentication of the consumer. The CAVV field is required for network liability shift, along with the XID and ECI fields. If any of these three fields is empty, the liability shift cannot be obtained. The CAVV must be base64 encoded.
currencycode	The ISO 4217, Currency Names and Code Elements, used to specify the amount submitted. These values are numeric and consist of three digits.
cvv2	The Card Verification Value (version 2) found on the back of the card. This value is not stored any longer than the length of the transaction. Review the config.properties file for valid character limitations under the hack character id *hc.bad_key*.
cvv2response	The response code returned from the host indicating the validity of the submitted CVV2 value.

Field Name	Description
eci	The electronic commerce indicator. This field is a network specific value which correlates to the success or failure of the transaction. This value is different based on the card type and brand. On requests, the ECI field represents the ECI returned from the ACS and is required for network liability shift along with the XID and CAVV fields. If any of these three fields is empty, the liability shift cannot be obtained.
error_code_tag	The internal ACI Commerce Gateway error code returned from the transaction request.
error_service_tag	The internal ACI Commerce Gateway service (if provided) that originated the error code.
error_text	A server-localized textual string explaining the meaning of the error_code_tag value if available.
errorurl	A valid URL specifying the destination to redirect a consumer to should an error occur during authentication or collection of payment details.
expday	Expiration day of the card. Assumed to be 1 if not specified.
expmonth	Expiration month of the card. Required.
expyear	Expiration year of the card. Required.
id	The TranPortal ID used to identify the terminal that will be used to validate and accept the transaction.
langid	The language ID (for the ACI Commerce Gateway configured language record) that identifies the localization to be used for presenting the hosted payment page.
member	Card member name. The name as it appears on the card.
merchantcert	Currently unused.
pareq	3-D Secure Payer Authentication Request value. This value is used by a 3-D Secure Access Control Server (ACS) to specify required values not previously made available by the Verify Enrollment Request.
pares	3-D Secure Payer Authentication Response value. This value is used to log the results of consumer authentication at the ACS.

Field Name	Description
password	The TranPortal password value. This value is being phased out as of ACI Commerce Gateway release 3.2. Either password or passwordhash may be used but passwordhash overrides the password.
passwordhash	The SHA1 hash representation of the TranPortal password. Beginning with ACI Commerce Gateway release 3.2, this will be the password authentication value to use with the Universal Servlet. Either password or passwordhash may be used but passwordhash overrides the password.
paymentid	Unique ID generated by and used within the ACI Commerce Gateway to identify a payment.
paymentpage	A valid URL specifying the destination to redirect a consumer to initiate the hosted payment page.
postdate	The host-specified posting date that will be used for the current transaction.
ref	Unique number generated and assigned by the ACI Commerce Gateway to a transaction going to the host. This number is used to track chargebacks back to ACI Commerce Gateway transactions.
responsecode	The host authorization response code from which the result code is calculated.
responseurl	A valid URL to which the merchant wants the ACI Commerce Gateway to reply for Merchant Notification after successful processing.

Field Name	Description
result	The ACI Commerce Gateway string representing the result code, based on the host response code. Valid responses: CAPTURED - The transaction was captured. APPROVED - The transaction was approved. VOIDED - The transaction was voided. NOT CAPTURED - The transaction was not captured. NOT APPROVED - The transaction was not approved. NOT VOIDED - The transaction was not voided. DENIED BY RISK - Risk denied the transaction. FAILED AVS - The transaction did not pass address verification. HOST TIMEOUT - The authorization system did not respond within the timeout limit.
trackid	The merchant assigned tracking ID. This ID should be unique, although no validation for this is provided within the ACI Commerce Gateway.
tranid	The ACI Commerce Gateway assigned transaction ID for a single action.
transid	The ACI Commerce Gateway assigned transaction ID for a single action which was previously generated. This ID is used to indicate the original transaction from which a subsequent transaction is based.
type	Card type. Valid values: ACNT - Account Brands CC - Credit DC - Debit PINDBT - PIN Debit SV - Stored Value UCAF - Universal Cardholder Authentication Field VPAS - Visa Payer Authentication Service

Field Name	Description
udf1 udf2 udf3 udf4 udf5	User Defined Fields 1, 2, 3, 4, and 5. If any fields are present, they are archived in the TRANLOG table. This field may be used for custom data required to be passed to and from the ACI Commerce Gateway. Review the config.properties file for valid character limitations under the hack character ID *hc.bad_text*.
url	The URL of the 3-D Secure Access Control Server (ACS) to redirect the consumer to during an ***MPIVerifyEnrollment*** transaction.
xid	The transaction ID used by a 3-D Secure Access Control Server (ACS) during generation of the CAVV. The XID field is normally optional but is required for network liability shift along with the CAVV and ECI fields. If any of these three fields is empty, the liability shift cannot be obtained.
zip	The postal code or ZIP Code of the card billing address. Review the config.properties file for valid character limitations under the hack character ID *hc.bad_text*.

Universal Plug-in Implementation Differences

Not all methods can be implemented in identical fashion across the currently supported runtimes and languages. This section details subtle differences in installation and implementation that need to be explained.

PHP - The PHP Version of the Universal Plug-in

- Requires newer PHP versions. During testing, version 5.2.3 performed well, while older versions, such as 5.1.6, generated errors.
- Contains an additional method ***setTimeout*** which may be used to set both the connect timeout and read timeout while communicating with the ACI Commerce Gateway.

C++ - The COM+ Version of the Universal Plug-in

- Requires that Microsoft XML version 4 (MSXML4) be installed with at least service pack 2. The COM+ implementation requires this library to be present.
- Has the ability to deliver asynchronous status messages during transactions. Any object that wants to receive these notifications must implement the IUniversalPluginStatus interface. The interface described for all of the ***get*** methods is using the IQuery Interface. If a different interface is needed, reference the associated .tld, .h, and idl files which are included in the Plugin Download that can be obtained from the ACI Commerce Gateway Merchant Website under the Developers menu.
- Requires that WinHTTP 2.0 or above be installed. The COM+ plug-in supports the following WINHTTP errors:
  - ERROR_WINHTTP_TIMEOUT
  - ERROR_WINHTTP_INTERNAL_ERRORR
  - ERROR_WINHTTP_INVALID_URL
  - ERROR_WINHTTP_NAME_NOT_RESOLVED
  - ERROR_WINHTTP_CANNOT_CONNECT
  - ERROR_WINHTTP_CONNECTION_ABORTED
- Contains an additional method ***setTimeout*** which may be used to set both the connect timeout and read timeout while communicating with the ACI Commerce Gateway.

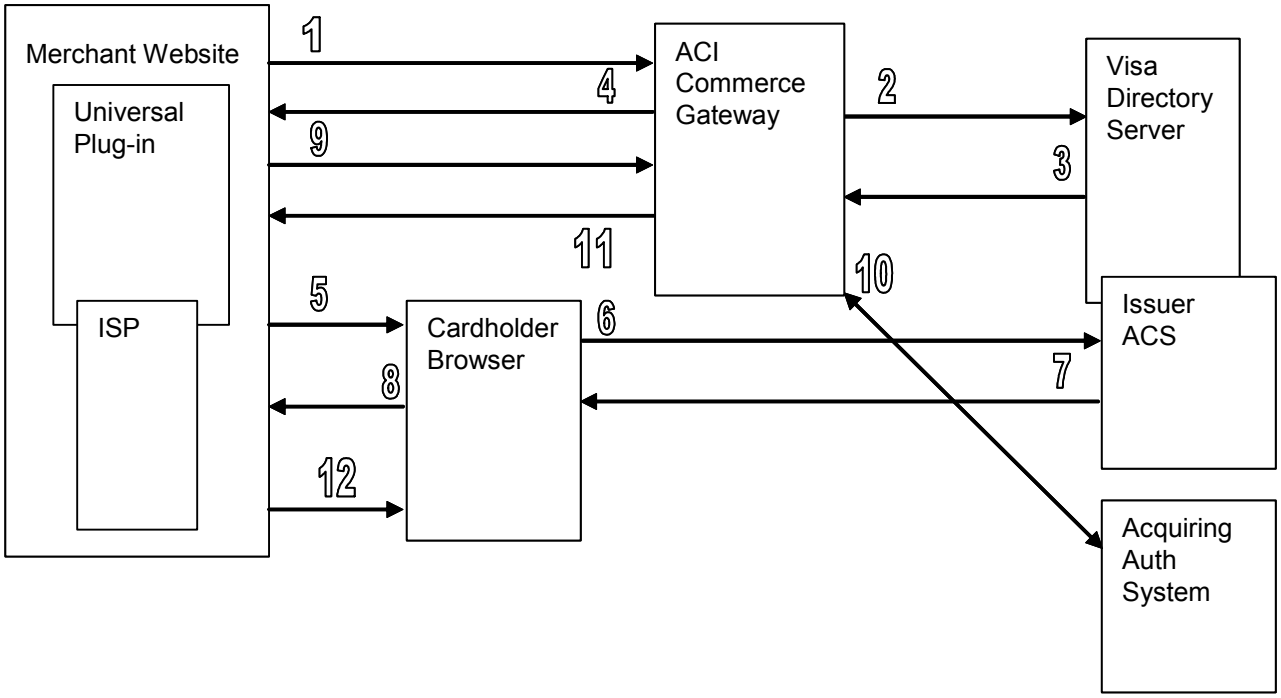
- The AppID for the COM+ version of the Universal Plug-in is: UniversalPlugin.UniversalPluginCtl while the GUID for the control is {2CE2AB54-4E7F-43C4-A3C4-E7205D0677C9}.

Java - The Java Version of the Universal Plug-in

- Currently does not support a standard way of controlling the connection and read timeouts. However, JVM support for controlling these timeouts is available through the sun.net.client.defaultConnectTimeout and sun.net.client.defaultReadTimeout JVM system properties.
- Generates critical errors via the NotEnoughDataException class.

A: 3-D Secure Merchant Payer Interface (MPI) Transaction Flow

The following diagram illustrates the 3-D Secure transaction flow using the MPIVerifyEnrollment and MPIPayerAuthentication transactions through the Universal Plug-in.

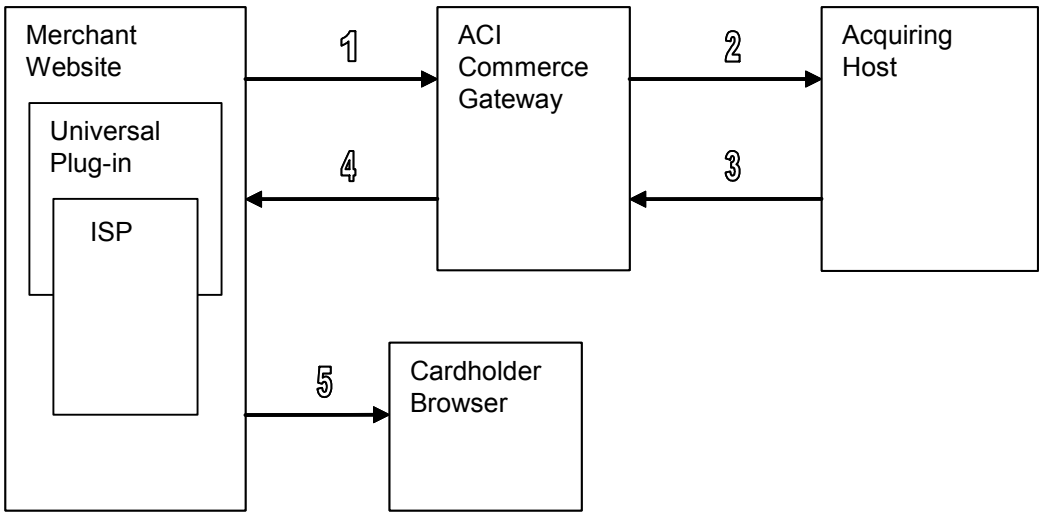


1. The merchant's system collects all of the payment data and sends it to the ACI Commerce Gateway, using the MPIVerifyEnrollment transaction through the Universal Plug-in.
2. The ACI Commerce Gateway builds and sends the VEReq to the Visa Directory Server using the 3-D Secure brand's endpoint.
3. The Visa Directory Server responds with a VERes to the ACI Commerce Gateway.

4. The ACI Commerce Gateway responds to the merchant's system using the Universal Plug-in and servlet, passing a properly formatted PAReq and an Access Control Server (ACS) URL to redirect the consumer to if the VERes was successful and the result was "ENROLLED." If the result is "NOT ENROLLED," ACI Commerce Gateway processes the transaction as normal using the VERes data. Continue with step 10.
5. The merchant's system redirects the cardholder to the issuer's ACS for the authentication.
6. The cardholder browser is redirected to the ACS together with the PAReq.
7. The ACS responds with a PARes to the cardholder browser.
8. The cardholder browser is redirected to the merchant's system with the PARes.
9. The merchant's system collects the PARes from the ACS and sends the received PARes using the MPIPayerAuthentication transaction using the Universal Plug-in to validate the PARes and to extract the data elements, such as ECI, XID, and CAVV.
10. The ACI Commerce Gateway validates the VERes/PARes, extracts the required data for the authorizing system, and receives a response.
11. The Authorization system responds to the merchant's system via the ACI Commerce Gateway with authorized data.
12. The merchant's system responds to the cardholder.

B: 3-D Secure Simple Transaction Flow

The following diagram illustrates the 3-D Secure transaction flow using the simple transactions, CardManagement and TranPortal (as well as the PaymentTran transaction for subsequent transactions) through the Universal Plug-in.

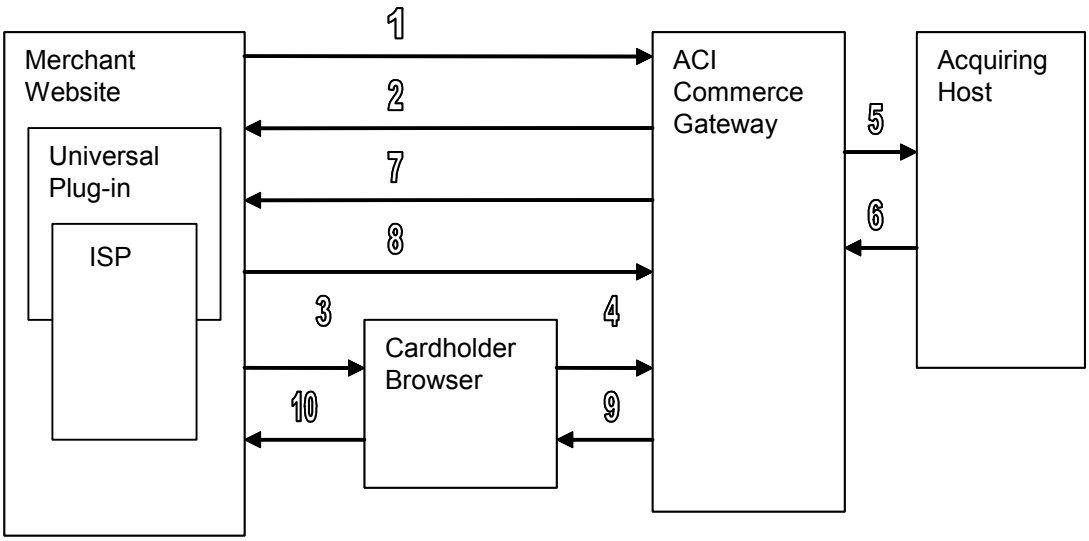


1. The merchant's system collects all of the data for the CardManagement and TranPortal transactions; or, in the case of subsequent transactions on a payment, the PaymentTran transaction, and sends it to the ACI Commerce Gateway, using the Universal Plug-in.
2. The ACI Commerce Gateway validates, formats, and sends the data to the host referenced by the endpoint configured on the terminal.
3. The Host responds to the ACI Commerce Gateway.
4. The ACI Commerce Gateway responds to the merchant's system using the Universal Plug-in.
5. The merchant's system presents the results to the consumer.

ACI Worldwide, Inc.

C: 3-D Secure Payment Transaction Flow

The following diagram illustrates the 3-D Secure transaction flow to make a purchase using the payment transactions, PaymentInit and PaymentTran, through the Universal Plug-in.



1. The merchant's system sends collected consumer and generated merchant data to the ACI Commerce Gateway, using the Universal Plug-in.
2. The ACI Commerce Gateway generates a payment record, stores the submitted data, sends a consumer redirect URL and payment id, and returns this as the token that the ACI Commerce Gateway will use to track this payment.
3. The merchant's system redirects the cardholder to the ACI Commerce Gateway consumer redirect URL, providing the payment id.
4. The cardholder is redirected to the ACI Commerce Gateway consumer URL and provides payment instrument information.

5. The ACI Commerce Gateway accepts the payment instrument data and updates the record created in step 2. The ACI Commerce Gateway validates, formats, and sends the collected payment data to the host referenced by the endpoint configured on the terminal.
6. The Host responds to the ACI Commerce Gateway.
7. The ACI Commerce Gateway notifies the merchant site (responseURL) of the results of the transaction.
8. The merchant notification (responseURL) reads transaction results and determines where on the merchant site to redirect the cardholder. The notification page or servlets returns one line, without any carriage returns containing: REDIRECT=URL. The URL specifies the merchant URL to which to redirect the cardholder.
9. The ACI Commerce Gateway redirects the cardholder to the URL provided in step 7.
10. The merchant's URL presents the results to the consumer.

D: C++ and Java Components for the BatchPortal Plug-in

C++ Components

---

class **Ie24BatchPipe**

Purpose:
This is a pure virtual interface. It is implemented by the C++ e24BatchPipe component. It provides methods for accessing the setup parameters and results as properties of the object. An instance of the e24BatchPipe C++ component is created using the factory method named Createe24BatchPipe. An instance of this object cannot be created using the new operator. Likewise, the delete operator cannot be used to delete an instance of the object. Instead, call the Release method to delete an instance of the object.

---

class **Ie24BatchPipeStatus**

Purpose:
This is a pure virtual interface. Any class that wants asynchronous status notifications from the e24BatchPipe C++ component must derive from this class and implement its methods.

---

char* GetAction();

Purpose:
Get the value of the ACTION object property.

---

char* GetBatchId();

Purpose:
Get the value of the BATCHID object property.

---

char* GetBatchTrackId();

Purpose:
Get the value of the TRACKID object property.

---

char* GetDataFile();

Purpose:
Get the value of the BATCHDATAFILE object property.

---

char* GetErrorMsg();

Purpose:
Get the value of the ErrorMsg object property.

---

char* GetOutData();

Purpose:
Get the value of the OUTDATA object property.

---

char* GetRawResponse();

Purpose:
Get the value of the RawResponse object property.

---

char* GetStatus();

Purpose:
Get the value of the STATUS object property.

---

char* GetStatusPercent();

Purpose:
Get the value of the STATUSPERCENT object property.

---

function Createe24BatchPipe

Purpose:
This is the factory function used to create an instance of the e24BatchPipe C++ component. This is the only way to create the component. The new operator cannot be used to create an instance of the component.

Definition:
Ie24BatchPipe* Createe24BatchPipe();

---

short PerformTransaction(Ie24BatchPipeStatus* pStatus);

Purpose:
Performs the transaction.

Parameters:
pStatus - A pointer to the Ie24BatchPipeStatus object that will receive asynchronous status messages. If NULL is used, then no asynchronous status messages will be delivered.

Returns:
0 if the transaction was successful; -1 otherwise.

---

typedef SBFACTORY

Purpose:
This is the typedef of the function pointer to the Createe24BatchPipe function. It can be used by applications that load this DLL dynamically using LoadLibrary, and get the function pointer using GetProcAddress.

Definition:
typedef Ie24BatchPipe* (WINAPI *SBFACTORY)();

---

virtual void OnStatusUpdate(const char* pMsg);

Purpose:
This is the method that is called on an object that receives asynchronous status messages during a transaction.

Parameters:
pMsg - a character string pointer that contains the message.

---

void SetAction(LPCSTR action);

Purpose:
Set the value of the ACTION object property.

Parameters:
action - A pointer to a character string that contains the new property value.

---

void SetAlias(LPCSTR alias);

Purpose:
Set the value of the ALIAS object property.

Parameters:
alias - A pointer to a character string that contains the new property value.

---

void SetBatchTrackId(LPCSTR trackid);

Purpose:
Set the value of the BATCHTRACKID object property.

Parameters:
trackid - A pointer to a character string that contains the new property value.

---

void SetDataFile(LPCSTR datafile);

Purpose:
Set the value of the BATCHDATAFILE object property. The DATAFILE property contains the full path to the local file that contains the batch data to be processed if the ACTION property equals 1. It contains the file that will contain the batch processing results if the ACTION property equals 2.

Parameters:
datafile - A pointer to a character string that contains the new property value.

---

void SetResourcePath(LPCSTR resourcepath);

Purpose:
Set the value of the resource path object property.

Parameters:
resourcepath - A pointer to a character string that contains the new property value.

---

void SetTimeout(long timeout);

Purpose:
Set the value of the TIMEOUT object property.

Parameters:
timeout - A pointer to a character string that contains the new timeout value.

Java Components

Class Hierarchy	class java.lang.Object class **e24BatchPipe**
Class e24BatchPipe	java.lang.Object | +--**e24BatchPipe** public class **e24BatchPipe** extends java.lang.Object
Constructor Summary	e24BatchPipe()

Method Summary

Following is the e24BatchPipe Java Class Object Method Summary documentation.

Return Type	Java Method and Description
java.lang.String	**getAlias**() Get the terminal resource alias.
java.lang.String	**getBatchAction**() Get the batch action value.
java.lang.String	**getBatchData**() Get the batch data.
java.lang.String	**getBatchDataFile**() Get the batch data file path and name.
java.lang.String	**getBatchId**() Get the batch ID.
java.lang.String	**getBatchTrackId**() Get the batch track ID value.
java.lang.String	**getDebugMsg**() Get the debug messages.

Return Type	Java Method and Description
boolean	**getDebugOn**() Get the boolean value whether debugging is enabled or not.
java.lang.String	**getError**() Get the error if on exists.
java.lang.String	**getFileSize**() Get the batch file size.
java.lang.String	**getResourcePath**() Get the file location of the resource file.
java.lang.String	**getStatus**() Get the batch status.
java.lang.String	**getStatusPercent**() Get the percent complete of the batch.
boolean	**performTransaction**() This is the main method.
void	**setAlias**(java.lang.String alias) Set the terminal alias.
void	**setBatchAction**(java.lang.String action) Set the batch action value.
void	**setBatchData**(java.lang.String data) Set the batch data value.
void	**setBatchDataFile**(java.lang.String file) Set the batch data file value.
void	**setBatchTrackId**(java.lang.String tid) Set the batch track ID value.
void	**setDebug**(boolean) Set the boolean vale to enable or disable debugging.
void	**setResourcePath**(java.lang.String resourcepath) Set the resource path for the ACI Commerce Gateway server.

Methods Inherited from Class java.lang.Object

- Clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

Constructor Detail

e24BatchPipe

public **e24BatchPipe()**

Method Detail

setResourcePath

---

public void **setResourcePath**(java.lang.String resourcepath)

Set the resource path for the ACI Commerce Gateway server.

Parameters:

resourcepath - The resource path of the resource file.

---

setAlias

public void setAlias(java.lang.String alias)

Set the terminal alias.

Parameters:

alias - The terminal alias value.

---

getBatchAction

public java.lang.String **getBatchAction**()

Get the batch action value.

Returns:

The batch action.

---

setBatchAction

public void **setBatchAction**(java.lang.String action)

Set the batch action value.

Parameters:

action - The password.

---

getBatchTrackId

public java.lang.String **getBatchTrackId**()

Get the batch track ID value.

Returns:

The batch track ID.

---

setBatchTrackId

public void **setBatchTrackId**(java.lang.String tid)

Set the batch track ID value.

Parameters:

tid - The batch track ID.

---

getBatchData

public java.lang.String **getBatchData**()

Get the batch data. This includes all the transactions for a given batch.

Returns:

The batch data.

---

setBatchData

public void **setBatchData**(java.lang.String data)

Set the batch data value. This includes all the transactions for a given batch.

Parameters:

data - The batch data.

---

getBatchDataFile

public java.lang.String **getBatchDataFile**()

Get the batch data file path and name.

Returns:

The batch data file string.

---

setBatchDataFile

public void **setBatchDataFile**(java.lang.String file)

Set the batch data file value.

Parameters:

file - The path and file name of the batch data file.

---

getFileSize

public java.lang.String **getFileSize**()

Get the batch file size.

Returns:

The file size.

---

getError

public java.lang.String **getError**()

Get the error if one exists.

Returns:

The error message, or null if there was no error.

---

getBatchId

public java.lang.String **getBatchId**()

Get the batch ID.

Returns:

The batch ID.

---

getStatus

public java.lang.String **getStatus**()

Get the batch status.

Returns:

The batch status.

---

getStatusPercent

public java.lang.String **getStatusPercent**()

Get the percent complete of the batch.

Returns:

The percent complete.

---

performTransaction

public boolean **performTransaction**()

throws java.lang.Exception

This is the main method. Calling this method from another class will initiate the e24BatchPipe. Prior to calling this method, you must set all the required fields using the ***gets*** and ***sets***.

Returns:

true if successful; false if an error occurred.

Throws:

java.lang.Exception - if an exception occurred.

Class NotEnoughDataException

java.lang.Object

|

+--java.lang.Throwable

|

+--java.lang.Exception

|

+--NotEnoughDataException

All Implemented Interfaces:

java.io.Serializable

---

public class **NotEnoughDataException**

extends java.lang.Exception

This exception is thrown if ProcessTransaction method of the SecureCharge class is invoked without setting the values of any properties.

See Also:

- Serialized Form

Constructor Summary

NotEnoughDataException()

The empty constructor.

NotEnoughDataException(java.lang.String msg)

The constructor with info.

Methods Inherited from class java.lang.Throwable

- fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

---

Methods Inherited from class java.lang.Object

- clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

---

Constructor Detail

NotEnoughDataException

public NotEnoughDataException()

The empty constructor.

NotEnoughDataException

public **NotEnoughDataException**(java.lang.String msg)

The constructor with info.

Parameters:

msg - The text message that describes the exception.